ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 4

«NP полные алгоритмы (Задача о разделении множеств)»

Выполнил работу

Исмагилова Юлия

Академическая группа №J3110

Принято

практик, Вершинин Владислав

Санкт-Петербург

2024

Содержание

[Содержание 2](#_Toc187271621)

[Введение 3](#_Toc187271622)

[Теоретическая подготовка 4](#_Toc187271623)

[Реализация 5](#_Toc187271624)

[Экспериментальная часть 7](#_Toc187271625)

[Заключение 9](#_Toc187271626)

[Приложения 10](#_Toc187271627)

Введение

**Цель работы:** разработать алгоритм на языке С++ для решения следующего технического задания: "Дан массив, необходимо разделить его на две части с равной суммой." Алгоритм должен решать задачу комбинаторным способом, сложность алгоритма должна быть не меньше, чем O(2^N). В массиве содержатся только положительные числа, максимум 25 элементов и сумма до 200.

**Задачи:**

1. Написать функцию, которая проверяет, возможно ли разделить массив на две части с равной суммой.
2. Написать функцию, которая находит индекс, на котором массив можно разделить на две части с равной суммой.
3. Написать функцию, которая разделяет массив на две части с равной суммой и возвращает два новых массива.

Теоретическая подготовка

**Типы данных**

1. Массив: содержит целые числа, которые необходимо разделить на две части с равной суммой.
2. Вектор: использован для динамического хранения элементов массива.
3. Библиотека numeric: предоставляет функции для выполнения числовых операций.

**Алгоритмы**

1. Проверка возможности разделения: для проверки возможности разделения массива на две части с равной суммой можно использовать алгоритм динамического программирования. Основная идея заключается в том, чтобы проверить, можно ли разделить массив на два подмножества с равной суммой.
2. Нахождение точки разделения: если массив можно разделить на две части с равной суммой, необходимо найти индекс, на котором это разделение возможно.
3. Разделение массива: после нахождения точки разделения, массив можно разделить на две части и вернуть два новых массива.

**Динамическое программирование**

Динамическое программирование (DP) — это метод решения задач, который использует предыдущие результаты для вычисления новых результатов. В данном случае DP можно использовать для проверки возможности разделения массива на две части с равной суммой.

Реализация

**Используемые библиотеки:**

1. <iostream>: для ввода и вывода данных.
2. <vector>: для работы с динамическими массивами.
3. <numeric>: для выполнения числовых операций, таких как суммирование элементов массива. (рис. 1)

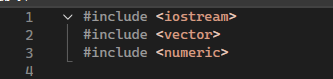


Рисунок 1 – пример использования в коде

**Partitioning\_check:**

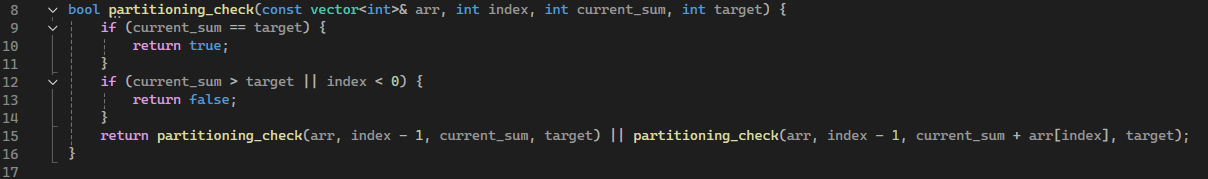
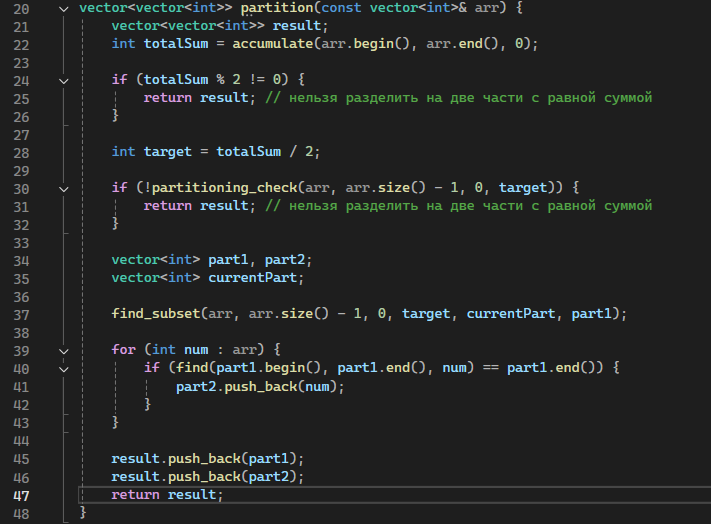
Реализация функции, которая проверяет, возможно ли разделить массив на две части с равной суммой. Функция использует рекурсивный подход для проверки возможности разделения массива: если текущая сумма равна целевой сумме, возвращается true, если превышает или индекс меньше 0 - false. В противном случае, функция рекурсивно проверяет два варианта: включение текущего элемента в сумму и исключение текущего элемента из суммы. (рис. 2)

Рисунок 2 – пример использования в коде

**Partition:**

Функция вычисляет общую сумму элементов массива: если общая сумма нечетная, разделение не может быть произведено. Если разделение возможно, функция вызывает find\_subset для нахождения подмножества с заданной суммой. (рис. 3)

Рисунок 3 – пример использования в коде

**Find\_subset****:**

Функция использует рекурсивный подход для нахождения подмножества с заданной суммой. Если текущая сумма равна целевой сумме, подмножество найдено, а если нет или индекс меньше 0, то возвращается **false**. (рис. 4)

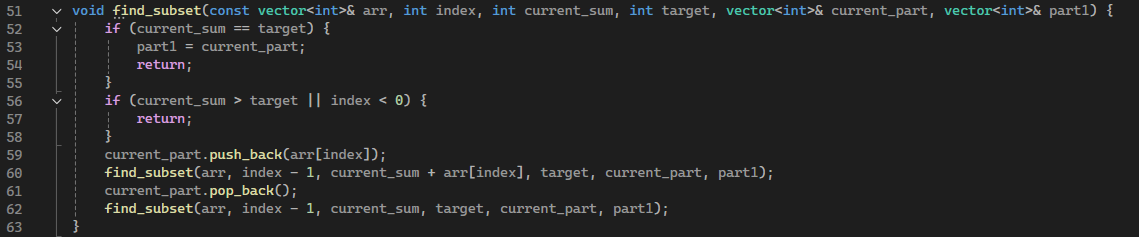


Рисунок 4 – пример использования в коде

Экспериментальная часть

**Подсчёт памяти:**

Объем памяти, занимаемый алгоритмом, будет зависеть от размера входного массива и глубины рекурсии. Примерный подсчёт памяти для прописанного теста:

1. Входной массив: 24 байта.
2. Результат: 72 байта.
3. Временные векторы: 36 байта.
4. Рекурсивные вызовы: 144 байта.

Итого: 24 + 72 + 36 + 144 = 276 байт.

**Подсчёт асимптотики:**

Алгоритм имеет временную сложность O(2n) и линейную пространственную сложность O(n).

1. Функция partitioning\_check:. временная сложность: O(2n), где n — количество элементов в массиве.
2. Функция partition: временная сложность: O(2n), так как она зависит от временной сложности partitioning\_check.
3. Функция find\_subset: временная сложность: O(2n).

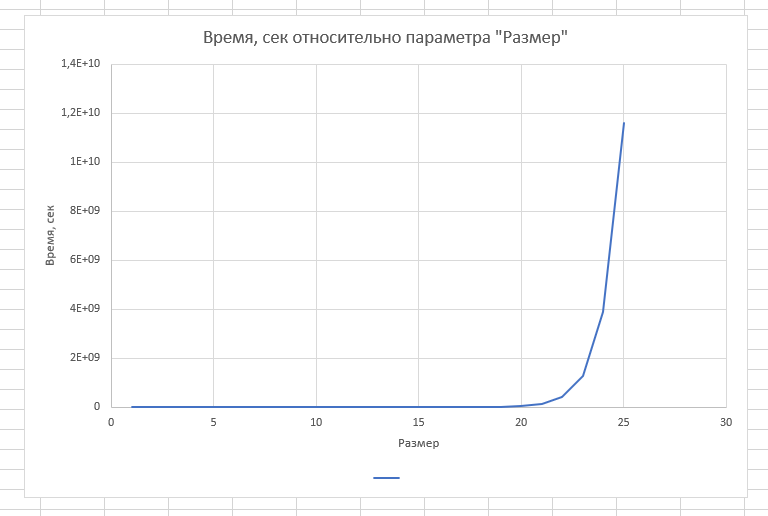
**График зависимости времени от числа элементов:**

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся до 25 элементов. Теоретически заданная сложность задачи составляет O(2^N) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1.

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора | 1 | 5 | 10 | 15 | 20 | 25 |
| Время выполнения программы, с | 0.001 | 5 | 810 | 196830 | 47829690 | 11622614670 |

График представляющий визуально удобный формат данных из таблицы №1 представлен на изображении №5.



Изображение №5 - График работы алгоритма

График показывает, что время выполнения алгоритма экспоненциально растет с увеличением размера массива. Это соответствует теоретической оценке временной сложности O(2n).

Заключение

В ходе выполнения работы мною был реализован алгоритм комбинаторного поиска для разделения массива на две части с равной суммой. Цель работы была достигнута путём тестирования на массивах с различным количеством элементов и различными суммами. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти, а также рассмотреть параллельные версии алгоритма для работы с большими массивами.

Приложения

ПРИЛОЖЕНИЕ А

Листинг кода:

#include <iostream>

#include <vector>

#include <numeric>

using namespace std;

bool partitioning\_check(const vector<int>& arr, int index, int current\_sum, int target) {

if (current\_sum == target) {

return true;

}

if (current\_sum > target || index < 0) {

return false;

}

return partitioning\_check(arr, index - 1, current\_sum, target) || partitioning\_check(arr, index - 1, current\_sum + arr[index], target);

}

// написали функцию для проверки, можно ли разделить массив на две части с равной суммой

vector<vector<int>> partition(const vector<int>& arr) {

vector<vector<int>> result;

int totalSum = accumulate(arr.begin(), arr.end(), 0);

if (totalSum % 2 != 0) {

return result; // нельзя разделить на две части с равной суммой

}

int target = totalSum / 2;

if (!partitioning\_check(arr, arr.size() - 1, 0, target)) {

return result; // нельзя разделить на две части с равной суммой

}

vector<int> part1, part2;

vector<int> currentPart;

find\_subset(arr, arr.size() - 1, 0, target, currentPart, part1);

for (int num : arr) {

if (find(part1.begin(), part1.end(), num) == part1.end()) {

part2.push\_back(num);

}

}

result.push\_back(part1);

result.push\_back(part2);

return result;

}

// написали функцию для разделения массива на две части с равной суммой

void find\_subset(const vector<int>& arr, int index, int current\_sum, int target, vector<int>& current\_part, vector<int>& part1) {

if (current\_sum == target) {

part1 = current\_part;

return;

}

if (current\_sum > target || index < 0) {

return;

}

current\_part.push\_back(arr[index]);

find\_subset(arr, index - 1, current\_sum + arr[index], target, current\_part, part1);

current\_part.pop\_back();

find\_subset(arr, index - 1, current\_sum, target, current\_part, part1);

}

// написали вложенную функцию для нахождения подмножества с заданной суммой

void test01() {

vector<int> arr = { 1, 5, 3, 5, 2, 5 };

vector<vector<int>> result = partition(arr);

if (!result.empty()) {

vector<int> Part1 = { 1, 5, 5 };

vector<int> Part2 = { 3, 2, 5 };

if (result[0] == Part1 && result[1] == Part2) {

cout << "Test1 passed" << endl;

}

else {

cout << "Test1 failed" << endl;

}

}

}

int main() {

setlocale(LC\_ALL, "rus");

test01();

return 0;

}